Base Language - Bug #2293

Fixing an unfixed extent output parameter causes an error

04/24/2014 05:21 PM - Hynek Cihlar

Status:	Closed	**Start date:**	04/24/2014
Priority:	Normal	**Due date:**	
Assignee:	Hynek Cihlar	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	Cleanup and Stablization for Server Features		
billable:	No	**case_num:**	
vendor_id:	GCD		

Description

Related issues:

Related to Base Language - Bug #2133: fix precision for decimal, dynamic-exte...	**Closed**	**01/21/2014**	**01/27/2014**
Related to Base Language - Bug #2292: Fix error handling when unfixed extent ...	**Closed**	**04/24/2014**	
Related to Base Language - Bug #2592: improve support for implicit type conve...	**Closed**		

History

#1 - 04/24/2014 05:29 PM - Hynek Cihlar

The following code causes the error *Indeterminate extent is already fixed to a dimension of 2* in P2J but works without error in Progress.

```
function foo returns int extent (output p as int extent).
  extent(p) = 2.
end.

def var i as int extent 2.
foo(i).
```

#2 - 04/24/2014 05:30 PM - Hynek Cihlar

- Subject changed from Fixing an unfixed extent input parameter causes an error to Fixing an unfixed extent output parameter causes an error

#3 - 04/25/2014 03:40 AM - Constantin Asofiei

Please post how the converted code for the program on note 1.

#4 - 04/25/2014 01:44 PM - Hynek Cihlar

- File FixingOutputParam.java added

The code sample in note 1 is missing parameter qualifier. Below is the corrected version.

```
function foo returns int extent (output p as int extent).
  extent(p) = 2.
end.

def var i as int extent 2.
```

```
foo(output i).
```

The converted code is attached.

#5 - 04/25/2014 01:50 PM - Constantin Asofiei

Hynek, this one is a little tricky: with output parameters, I think 4GL assumes nothing is received from the original argument - including its extent. Please check this:

1. what does the extent function report for the p parameter received by foo
2. what happens if you use extent(p) = 3 - i.e. something different than the argument's extent?

#6 - 04/25/2014 03:11 PM - Greg Shah

- Target version set to Milestone 11

#7 - 04/25/2014 05:49 PM - Hynek Cihlar

Constantin Asofiei wrote:

> Hynek, this one is a little tricky: with output parameters, I think 4GL assumes nothing is received from the original argument - including its extent. Please check this:
>
> 1. what does the extent function report for the p parameter received by foo

Here are a few variations:

Case 1:

```
function foo returns int extent (output p as int extent).
  message "extent(p) =" extent(p).
  extent(p) = 2.
  message "extent(p) =" extent(p).
end.

def var i as int extent 2.
foo(output i).
message "extent(i) =" extent(i).
```

outputs:

```
extent(p) = ?
extent(p) = 2
extent(i) = 2
```

Case 2:

```
function foo returns int extent (output p as int extent).
  message "extent(p) =" extent(p).
  extent(p) = 2.
  message "extent(p) =" extent(p).
end.
```

```
def var i as int extent.
extent(i) = 2.
foo(output i).
message "extent(i) =" extent(i).
```

outputs:

```
extent(p) = ?
extent(p) = 2
extent(i) = 2
```

Case 3:

```
function foo returns int extent (output p as int extent).
  message "extent(p) =" extent(p).
  extent(p) = 3.
  message "extent(p) =" extent(p).
end.

def var i as int extent.
extent(i) = 2.
foo(output i).
message "extent(i) =" extent(i).
```

outputs:

```
extent(p) = ?
extent(p) = 3
Extent parameter dimension of 2 from procedure /home/hc/p27581_Untitled4.ped is mismatched with sub-procedure
foo /home/hc/p27581_Untitled4.ped parameter
dimension of 3 ...(328) (11428)
extent(i) = 2
```

Case 4:

```
function foo returns int extent (output p as int extent).
  message "extent(p) =" extent(p).
  extent(p) = ?.
  message "extent(p) =" extent(p).
end.

def var i as int extent.
extent(i) = 2.
foo(output i).
message "extent(i) =" extent(i).
```

outputs:

```
extent(p) = ?
extent(p) = ?
Extent parameter dimension of 2 from procedure /home/hc/p43714_Untitled5.ped is mismatched with sub-procedure
foo /home/hc/p43714_Untitled5.ped parameter
dimension of 0 ...(328) (11428)
extent(i) = 2
```

1. what happens if you use extent(p) = 3 - i.e. something different than the argument's extent?

In this case error is raised and execution continues with the next statement after the function call: "Extent parameter dimension of 2 from procedure /home/hc/p51007_Untitled1.ped is mismatched with sub-procedure foo /home/hc/p51007_Untitled1.ped parameter dimension of 3 ...(328) (11428)". See **Case 3** above.

#8 - 04/25/2014 05:54 PM - Hynek Cihlar

- Status changed from New to WIP

#9 - 05/20/2014 05:27 AM - Constantin Asofiei

Hynek, something else to test: what happens with an extent var passed as an OUTPUT argument, if the function/proc does not change it at all? Cases to consider (try to cover as many variations as you can think of):
1. the var is dynamic extent or fixed extent
2. the var's elements are initialized or not

The above tests show that if the received extent var has its extent set, that can't be changed.

#10 - 05/21/2014 11:16 AM - Hynek Cihlar

Constantin Asofiei wrote:

> Hynek, something else to test: what happens with an extent var passed as an OUTPUT argument, if the function/proc does not change it at all? Cases to consider (try to cover as many variations as you can think of):
> 1. the var is dynamic extent or fixed extent
> 2. the var's elements are initialized or not

What happens with the var will depend on the runtime compatibility of the output argument and the variable. From the experiments I did it seems that Progress internally defines new variable for the output argument (based on the type info defined at the argument definition) and before the function block is exited it assigns it to the var originally passed in. These two variables seem to be disconnected during runtime and they are type-independent with one exception - fixing the argument variable with the extent statement checks the extent of the variable passed in and will issue an error when their extents don't match. I will illustrate the behavior on the following examples.

Case 1

```
function foo returns int extent (output p as int extent 2).
  message "extent(p) =" extent(p).
  p[2] = 9.
end.

def var i as int extent 2 init [3,4].
foo(output i).
message "extent(i) =" extent(i) "i[1,2] =" i[1] i[2].
```

outputs

```
extent(p) = 2
extent(i) = 2 i[1,2] = 0 9
```

Case 2

```
function foo returns int extent (output p as int extent).
  message "extent(p) =" extent(p).
```

```
  p[2] = 9.
end.

def var i as int extent 2 init [3,4].
foo(output i).
message "extent(i) =" extent(i) "i[1,2] =" i[1] i[2].
```

outputs

```
extent(p) = ?
Invalid assignment to an unfixed indeterminate extent (11389)
Extent parameter dimension of 2 from procedure /home/hc/p84008_Untitled1.ped is mismatched with sub-procedure
foo /home/hc/p84008_Untitled1.ped parameter dimension of
0 ...(328) (11428)
extent(i) = 2 i[1,2] = 3 4
```

Case 3

```
function foo returns int extent (output p as int extent 2).
  message "extent(p) =" extent(p).
end.

def var i as int extent 2 init [3,4].
foo(output i).
message "extent(i) =" extent(i) "i[1,2] =" i[1] i[2].
```

outputs

```
extent(p) = 2
extent(i) = 2 i[1,2] = 0 0
```

Case 4

```
function foo returns int extent (output p as int extent 3).
  message "extent(p) =" extent(p).
end.

def var i as int extent.
foo(output i).
message "extent(i) =" extent(i) "i[1,2] =" i[1] i[2].
```

outputs

```
extent(p) = 3
extent(i) = 3 i[1,2] = 0 0
```

Case 5

```
function foo returns int extent (output p as int extent 3).
  message "extent(p) =" extent(p).
end.

def var i as int extent.
extent(p) = 2.
foo(output i).
message "extent(i) =" extent(i) "i[1,2] =" i[1] i[2].
```

outputs

```
Extent parameter dimension of 2 from procedure /home/hc/p03040_Untitled4.ped is mismatched with sub-procedure
foo /home/hc/p03040_Untitled4.ped parameter dimension of 3 ...(328) (11428)
extent(i) = 2 i[1,2] = 0 0
```

Case 6

```
function foo returns int extent (output p as int extent).
  message "extent(p) =" extent(p).
  extent(p) = 3.
  message "extent(p) =" extent(p).
end.

def var i as int extent 2 init [3,4].
foo(output i).
message "extent(i) =" extent(i) "i[1,2] =" i[1] i[2].
```

outputs

```
extent(p) = ?
extent(p) = 3
Extent parameter dimension of 2 from procedure /home/hc/p03040_Untitled4.ped is mismatched with sub-procedure
foo /home/hc/p03040_Untitled4.ped parameter dimension of 3 ...(328) (11428)
```

I didn't find any differences in runtime behavior whether the variable being passed to a function was initialized in the var statement or with a simple assignment, or whether it was an indeterminate extent (fixed with the extent statement) or a determinate extent.

The solution I believe would be to change the conversion such that the output parameter variable in the function would not be initialized to the reference of the passed in variable (see OutputExtentParameter class), but new array instance would be created. Before the function exit, the output parameter would be assigned back to the variable passed in the function. The conversion would also have to implement the runtime check for the extent statement so that proper error is issued when incompatible extent is set on the parameter variable.

The above covers functions only. I will also test procedure output parameters to see how it behaves there and whether there are any differences in P2J.

#11 - 05/21/2014 01:30 PM - Constantin Asofiei

Hynek Cihlar wrote:

> The above covers functions only. I will also test procedure output parameters to see how it behaves there and whether there are any differences in P2J.

Please also check what happens if the function/proc ends abnormally (i.e. a RETURN ERROR or some other condition is raised).

#12 - 05/21/2014 06:02 PM - Hynek Cihlar

Constantin Asofiei wrote:

> Hynek Cihlar wrote:
>
> > The above covers functions only. I will also test procedure output parameters to see how it behaves there and whether there are any differences in P2J.
>
> Please also check what happens if the function/proc ends abnormally (i.e. a RETURN ERROR or some other condition is raised).

When error is raised during function or procedure execution, the variable passed in the block won't be assigned to the value set inside the block as can be seen from the sample below.

```
function foo returns int extent (output p as int extent 2).
  message "extent(p) =" extent(p).
  p[2] = 9.
  return error.
end.

def var i as int extent 2 init [3,7].
foo(output i).
message "extent(i) =" extent(i) "i[1,2] =" i[1] i[2].
```

outputs

```
extent(p) = 2
extent(i) = 2 i[1,2] = 3 7
```

#13 - 05/22/2014 01:41 AM - Constantin Asofiei

Hynek Cihlar wrote:

> When error is raised during function or procedure execution, the variable passed in the block won't be assigned to the value set inside the block as can be seen from the sample below.

OK, please confirm that the same is with the var's extent (i.e. an uninitialized extent remains uninitialized if the proc has RETURN ERROR, regardless of how the proc's parameter is defined - i.e. fixed extent - or if the var's extent is set explicitly in the proc's body).

After this, I think you have enough to work on the implementation. I suspect OutputExtentParameter will have to implement the Finalizable interface.

#14 - 05/22/2014 11:39 AM - Hynek Cihlar

Constantin Asofiei wrote:

> Hynek Cihlar wrote:
>
> > When error is raised during function or procedure execution, the variable passed in the block won't be assigned to the value set inside the block as can be seen from the sample below.
>
> OK, please confirm that the same is with the var's extent (i.e. an uninitialized extent remains uninitialized if the proc has RETURN ERROR, regardless of how the proc's parameter is defined - i.e. fixed extent - or if the var's extent is set explicitly in the proc's body).

Yes. this is confirmed.

> After this, I think you have enough to work on the implementation. I suspect OutputExtentParameter will have to implement the Finalizable interface.

Currently OutputExtentParameter uses another approach to assign the local variable back to the passed-in variable. An instance implementing OutputParameterAssigner interface is registered with TransactionManager. BlockManager then retrieves this reference and calls OutputParameterAssigner.processAssignments after it executes the function block and after it pops the current scope. To me this seems a little hacky when there are already other more generic notification mechanisms to which OutputExtentParameter can bind to, for example Finalizable as you mention.

#15 - 05/22/2014 01:20 PM - Hynek Cihlar

The implementation should be straightforward with one exception - fixing the indeterminate extent argument. When the argument is declared as indeterminate extent, Progress checks the value used in the extent statement. The value must match the extent value of the passed-in variable.

In terms of implementation, I don't have a better idea than storing the variable pair consisting of the passed-in variable and the argument variable and use the pair in ArrayAssigner when resizing (fixing) the extent array. There in the resize method the extent values would be compared and error would be issued if not matched. OutputExtentParameter could for example register the references with ArrayAssigner in a scoped instance so the references would be removed on scope popup.

#16 - 05/23/2014 01:50 AM - Constantin Asofiei

Hynek Cihlar wrote:

> In terms of implementation, I don't have a better idea than storing the variable pair consisting of the passed-in variable and the argument variable and use the pair in ArrayAssigner when resizing (fixing) the extent array.

This seems about right. How do you expect the converted code will look?

#17 - 05/23/2014 02:37 PM - Hynek Cihlar

Constantin Asofiei wrote:

> How do you expect the converted code will look?

There are not many changes in the converted code. The difference will be only in the allocation of output parameter. Here's the code, the changes are shown with code comments.

```
public integer[] foo(final OutputExtentParameter<integer> extp)
{
   return extentFunction(integer.class, new Block()
   {
      // changed line getVariable -> createParameter
      integer[] p = extp.createParameter();

      public void init()
      {
         extp.setParameter(p);
         assignMulti(p, new integer(0));
         ArrayAssigner.registerDynamicArray(p);
         TransactionManager.deregister(new Undoable[][]
         {
            p
         });
      }

      public void body()
      {
         message(new Object[]
         {
            "extent(p) =",
            ArrayAssigner.lengthOf(p)
         });
         p = ArrayAssigner.resize(p, 2);
```

```
            extp.setParameter(p);
            message(new Object[]
            {
               "extent(p) =",
               ArrayAssigner.lengthOf(p)
            });
            returnError();
         }
      });
   }
```

The main change in the runtime support to allow the above I did was renaming OutputExtentParameter to AbstractExtentParameter and creating two new subclasses classes OutputExtentParameter and InputOutputExtentParameter. OutputExtentParameter declares new method createParameter which takes care of creating new array instance and registering it with ArrayAssigner to enable check for fixing the array. Splitting the types has other advantages: (1) input-output and output parameters have different semantics for extent variable initialization, these different behaviors are now implemented in the subclasses. That means that the converted code always calls getVariable(int) regardless output or input-output and @getVariable(int, boolean) is not needed any more - this makes the initialization transparent to the converted code and the converted code is cleaner. (2) We are not loosing the information whether the parameter was declared as output or input-output.

#18 - 05/27/2014 11:59 AM - Constantin Asofiei

Hynek,

The approach loosk good, just a question about how the converted code looks: if we have integer[] p = extp.createParameter();, is the extp.setParameter(p); still needed?

#19 - 05/27/2014 01:59 PM - Hynek Cihlar

Constantin Asofiei wrote:

> Hynek,
>
> The approach loosk good, just a question about how the converted code looks: if we have integer[] p = extp.createParameter();, is the extp.setParameter(p); still needed?

Good question. Well I was assuming it was needed. The reasoning I took was that setParameter was technically not needed even before my change (with splitting getVariable into two methods to allow additional logic like calling setParameter during the getVariable call). Thus I assumed that it was beneficial for the editor to see the parameter setting, maybe for some additional custom logic. If this assumption is wrong I would gladly remove setParameter from the public interface (for both input and input-output) and handle it transparently to the converted code.

#20 - 05/28/2014 10:36 AM - Hynek Cihlar

By the way, we can remove the need to call setParameter from the converted code completely. For example, when extent(param) is issued for an input-output parameter, new array is created and new reference created must be put back to the parameter by calling setParameter. This can be handled transparently without any participation in the converted code.

#21 - 05/28/2014 11:45 AM - Hynek Cihlar

Is the conversion of output extent fields supposed to work? When converting a temp-table field passed as output parameter to a function, an OutputExtentField constructor call is emitted with incorrect parameters resulting in compilation error. Tested without my conversion changes.

#22 - 05/28/2014 11:52 AM - Constantin Asofiei

Hynek Cihlar wrote:

> Is the conversion of output extent fields supposed to work? When converting a temp-table field passed as output parameter to a function, an OutputExtentField constructor call is emitted with incorrect parameters resulting in compilation error. Tested without my conversion changes.

Not sure if you are aware, table fields can not be set as dynamic extent in 4GL. How does your code look like? If it compiles in 4GL, then this means P2J has a problem - in this case, create a separate task and place details there.

#23 - 05/28/2014 12:35 PM - Hynek Cihlar

Constantin Asofiei wrote:

> Not sure if you are aware, table fields can not be set as dynamic extent in 4GL. How does your code look like? If it compiles in 4GL, then this means P2J has a problem - in this case, create a separate task and place details there.

Here's the example.

```
def temp-table tt1 field f1 as decimal extent 5 field f2 as int.

function foo_field returns int (output p as decimal extent).
extent(p) = 5.
end.

create tt1.
find first tt1.
foo_field(output tt1.f1).
```

will emmit

```
   TempRecord1.Buf tt1 = TemporaryBuffer.define(TempRecord1.Buf.class, "tt1", "tt1", false);

   /**
    * External procedure (converted to Java from the 4GL source code
    * in fixing_output_field.p).
    */
   public void execute()
   {
      externalProcedure(new Block()
      {
         public void body()
```

```
        {
           RecordBuffer.openScope(tt1);
           tt1.create();
           new FindQuery(tt1, (String) null, null, "tt1.id asc").first();
           fooField(new OutputExtentField<decimal>(tt1)); // COMPILATION ERROR
        }
     });
  }
```

Note the compilation error at the OutputExtentField constructor call.

#24 - 05/28/2014 12:43 PM - Constantin Asofiei

Hynek Cihlar wrote:

> Constantin Asofiei wrote:
>
> > Not sure if you are aware, table fields can not be set as dynamic extent in 4GL. How does your code look like? If it compiles in 4GL, then this means P2J has a problem - in this case, create a separate task and place details there.
>
> Here's the example.

OK, this is a conversion problem, if the fix is not trivial leave it for another time.

#25 - 05/28/2014 02:38 PM - Hynek Cihlar

Constantin Asofiei wrote:

> OK, this is a conversion problem, if the fix is not trivial leave it for another time.

It is blocking the other changes I have in place and waiting to be tested so I'll give it a minute. Hopefully it will be nothing serious.

#26 - 06/13/2014 04:11 AM - Hynek Cihlar

In the constructor FieldReference(Database, Class<?>, String) the setter is not set and the sizer is. This is suspicious, I would expect vice versa - setter set, sizer not.

#27 - 06/13/2014 04:32 AM - Hynek Cihlar

Actually, setting the sizer in the constructor FieldReference(Database, Class, String) makes sense. So the question is, why the setter is not set.

#28 - 06/13/2014 10:21 AM - Greg Shah

The idea of the FieldReference is to allow the delegation of reading/writing a database field to runtime code that is not database aware. For example, when you edit a database field using something like the UPDATE statement, there is no "lvalue" reference for the database field like there would be for a variable. To solve this, we create the FieldReference which can both read and write to the database field that is configured at construction time.

The actual reading (and writing if necessary) is deferred until the runtime code needs to access it. In the UPDATE example, the setter would not be called unless a successful edit occurred and the modified changes were written back from the frame's screen-buffer into the FieldReference. At the time of construction, there are no changes to make yet and it is not even known if changes will be made.

#29 - 06/14/2014 10:49 AM - Hynek Cihlar

Greg Shah wrote:

> The actual reading (and writing if necessary) is deferred until the runtime code needs to access it. In the UPDATE example, the setter would not be called unless a successful edit occurred and the modified changes were written back from the frame's screen-buffer into the FieldReference. At the time of construction, there are no changes to make yet and it is not even known if changes will be made.

What I am trying to say is that if you construct the FieldReference with any "non-indexed" constructor (i.e. FieldReference(Database, Class<?>, String), the created instance cannot be used to write its field back, ever. The reason for this is that FieldReference.setter is not set in the constructor nor anywhere else for the created instance.

#30 - 06/14/2014 11:42 AM - Constantin Asofiei

Hynek Cihlar wrote:

> What I am trying to say is that if you construct the FieldReference with any "non-indexed" constructor (i.e. FieldReference(Database, Class<?>, String), the created instance cannot be used to write its field back, ever. The reason for this is that FieldReference.setter is not set in the constructor nor anywhere else for the created instance.

Look from where the c'tor is used:

```
DMOSorter$FieldWorker(Database, Class<?>, SortCriterion)
DynamicLegacyKeyJoin.setup(RecordBuffer, RecordBuffer, RelationInfo, boolean)
ForeignResolver.createFieldRefs(Class, Iterator)
```

This suggests that the FieldReference instance created via that c'tor is read-only, and is expected to be used only for retrieving the value, not changing it.

Eric: please confirm this.

#31 - 06/14/2014 04:48 PM - Eric Faulhaber

Constantin Asofiei wrote:

> This suggests that the FieldReference instance created via that c'tor is read-only, and is expected to be used only for retrieving the value, not changing it.
>
> Eric: please confirm this.

Yes, I recall the need at the time was for a read-only instance for this handful of uses. I didn't set the sizer initially; that was added with a later update, not sure why.

Hynek, if you are making edits in this class, please add a note to the javadoc that this c'tor is meant to create a read-only instance.

#32 - 06/16/2014 08:21 AM - Hynek Cihlar

I didn't find any write-usages myself and from the code it wasn't clear whether it was planned to use the constructors to write fields as well. I am doing changes in the file and I extended the Javadoc to mention the instance is intended to read-only.

#33 - 06/16/2014 11:24 AM - Greg Shah

Eric: is there a reason not to allow this to naturally support write use cases?

#34 - 06/16/2014 12:28 PM - Eric Faulhaber

Greg Shah wrote:

> Eric: is there a reason not to allow this to naturally support write use cases?

It's not needed and it adds unnecessary overhead (albeit very little -- but why add unnecessary overhead anywhere?).

IIRC, the original intent was to have a short-lived instance, with minimal overhead in the c'tor, for very specific (read-only, non-extent) use cases. I only needed the getter method. I'm not sure why the sizer initialization was added later (doesn't make sense for non-extent cases), unless it was to prevent a problem with some other, newer downstream code which added a dependency on the sizer being non-null. The original implementation worked without it.

#35 - 06/18/2014 10:01 AM - Hynek Cihlar

- File hc_upd20140618a.zip added

The attached file provides

- fix for output extent parameter dimension not initialized to unknown

```
function foo1 returns int extent (output p as int extent).
  if extent(p) <> ? then message "NOT EXPECTED".
end.

def var i as int extent 2.
foo1(output i).
```

- fix for output unfixed dynamic extent parameter cannot be fixed

```
function foo1 returns int extent (output p as int extent).
  /* FAILS ON THE NEXT LINE WITH: Indeterminate extent is already fixed to a dimension 2. */
  /* No error is expected. */
  extent(p) = 2.
end.

def var i as int extent 2.
foo1(output i).
```

- fix for input-output and output parameter should not yield decimal precision

```
function foo11 returns int extent (output p as decimal extent).
  extent(p) = 2.
  p[1] = 1.5555555555.
end.

def var i15 as decimal decimals 3 extent 2.
foo11(output i15).
if i[1] <> 1.5555555555 then message "NOT EXPECTED".
```

- fix for raising an error in a function block is expected to revert value changes for output and input-output parameters
  There is an open issue for this fix, see below.

```
function foo8 returns int extent (input-output p as int extent 2).
  p[2] = 9.
  return error.
end.

def var i12 as int extent 2 init [3,7].
foo8(input-output i12).
if i12[1] <> 3 then message "NOT EXPECTED".
if i12[2] <> 7 then message "NOT EXPECTED".

function foo9 returns int extent (output p as int extent 2).
  return error.
end.

def var i13 as int extent.
foo9(output i13).
if extent(i13) <> ? then message "NOT EXPECTED".
```

- Fixes for decimal handling of the extent parameter references and of the target variable assignment. This is covered by #2295.
- Extent parameter wrapper classes improvements - the change set simplifies the interface of the extent parameter wrapper classes and hides the initialization logic and ArrayAssigner registration logic as well as automates parameter updates on reference changes.
- Fixes for several conversion and runtime issues of an extent field passed in to an input-output or output parameter.
  Functional testing for this is in progress.

There are a few open issues

1. OutputExtentParameter.validateAssignment raises an error of "Extent parameter dimension of %d from procedure %s is mismatched with...". I haven't figured out how to retrieve the sub-procedure name.

2. In the converted code, the procedure calls are generated with ErrorManager.silentErrorEnable(). This prevents some errors which are expected in some use cases, like 'Extent parameter dimension of 2 from procedure...'.

```
            ErrorManager.silentErrorEnable();
```

```
            ControlFlowOps.invokeWithMode("proc", "O", new ExtentExpr14());
            ErrorManager.silentErrorDisable();
            if (_isNotEqual(i19.length, 2))
            {
               message("ERR i19 <> 2");
            }
```

3. How to detect ERROR RETURN? BlockManager.returnError doesn't seem to set a pending error and then AbstractExtentParam.assign performs the variable assignment when it should not.

Please review the code and comment on the issues above.

#36 - 06/18/2014 10:19 AM - Hynek Cihlar

I have also checked-in all test cases for the above changes. See testcases/uast/output_var_param.p and testcases/uast/input_output_var_param.p.

#37 - 06/19/2014 04:41 AM - Constantin Asofiei

Review for 0618a.zip:

- fix the copyright date in annotations/output_parameters.rules and convert/output_parameters.rules.
- ConversionDriver - remove the "TODO:" prefix, as it's no longer needed (the logging was addressed).
- FieldReference: the Note that the created instance can be used to read the related field only. needs to be added only to the FieldReference(Database database, Class<?> dmoIface, String property) c'tor: the other c'tors are not read-only.
- FieldReference.getGetter and getSetter and the changes in FieldReference c'tor (where these are called): I guess you are fixing a latent bug there, right?
- FieldReference.fetchExtentMethods - there is something missing in your code. You should check if the property is really extent or not, before checking if the DMO method is indexed. Also, note that there is more than one indexed DMO method for a property - how do you handle this? Is it OK to use only the one which receives the index as a native int? Also, fetchExtentMethods I think is better suited in the PropertyHelper class.
- InputOutputExtentField
  - the imports need to use the asterix.
  - formatting for the c'tor is not right - the parameters need to be left-aligned.
  - InputOutputExtentField(FieldReference) is missing javadoc
- OutputExtentField(FieldReference) is missing javadoc
- AbstractExtentParameter
  - the header and class javadoc exceed the max line length (98)
  - if the getVariable APIs are not needed, remove them
  - assign - why not throw a NullPointerException? Also, the message at line 194 has problems: you have tabs and the starting quote is in the wrong place.
  - javadoc lines 217 and 234 have link out-of-place.

About your questions:

> 1. OutputExtentParameter.validateAssignment raises an error of "Extent parameter dimension of %d from procedure %s is mismatched with...". I haven't figured out how to retrieve the sub-procedure name.

This is a tricky one. The assignment takes place AFTER the procedure/function has already finished - thus it is no longer on the stack. I think you need to modify the APIs (starting with OutputParameterAssigner.processAssignments to pass as parameter the function/internal proc name and the external program relative name (as returned by ProcedureManager.getRelativeName).

> 2. In the converted code, the procedure calls are generated with ErrorManager.silentErrorEnable(). This prevents some errors which are expected in some use cases, like 'Extent parameter dimension of 2 from procedure...'.

Please tell me which program I can look at, to test this. In normal circumstances, the NO-ERROR clause (which is the reason why the ControlFlowOps.invoke call is enclosed in ErrorManager.silentErrorEnable/Disable) will prevent the enclosed code to raise/show errors. But there might be some exception in this case.

> 3. How to detect ERROR RETURN? BlockManager.returnError doesn't seem to set a pending error and then AbstractExtentParam.assign performs the variable assignment when it should not.

Isn't RETURN ERROR the same as raising a condition in the procedure/function? If so, then I think if you catch the ErrorConditionException (or catch RuntimeException and check the causes), you can assume the proc/function did not end properly. If this doesn't work, you will need to set some flag in BlockManager with the state of how the last func/proc call ended - via normally, via return error, with a condition, etc.

#38 - 06/19/2014 06:27 PM - Hynek Cihlar

- File silent_error.p added

- File hc_upd20140619a.zip added

- File SilentError.java added

Constantin Asofiei wrote:

> Review for 0618a.zip:
>
> - fix the copyright date in annotations/output_parameters.rules and convert/output_parameters.rules.
> - ConversionDriver - remove the "TODO:" prefix, as it's no longer needed (the logging was addressed).
> - FieldReference: the Note that the created instance can be used to read the related field only. needs to be added only to the FieldReference(Database database, Class<?> dmoIface, String property) c'tor: the other c'tors are not read-only.

Fixed.

> - FieldReference.getGetter and getSetter and the changes in FieldReference c'tor (where these are called): I guess you are fixing a latent bug there, right?

Well, the bug was latent as long as you haven't tried to use fields as output parameters for example. FieldReference used to operate only with the indexed property methods (taking the primitive integer type). After the fix, the proper set of methods is used depending whether the reference is created as indexed or nonindexed.

> - FieldReference.fetchExtentMethods - there is something missing in your code. You should check if the property is really extent or not, before checking if the DMO method is indexed. Also, note that there is more than one indexed DMO method for a property - how do you handle this? Is it OK to use only the one which receives the index as a native int? Also, fetchExtentMethods I think is better suited in the PropertyHelper class.

Good questions.

I think the method identifier is not chosen well - the method is capable to operate on extent and non-extent properties. It should have been named something like fetchIndexedMethods.

Yes, I am aware that there are multiple signatures of the indexed methods. I am only utilizing the primitive-int ones as that seemed to had been the intention of the original code. And I think it is ok to use only the int signature, as it seems the other ones are there only for type compatibility.

Originally I indeed attempted to resolve the problem in PropertyHelper, but later on I ditched the code and moved it up to FieldReference. I was afraid

I would not be able to test and fix all the possible regressions I surely introduced. It seems there are multiple places that rely on PropertyHelper's specific behavior of preferring indexed methods over their non-indexed counterparts.

- InputOutputExtentField
  - the imports need to use the asterix.
  - formatting for the c'tor is not right - the parameters need to be left-aligned.
  - InputOutputExtentField(FieldReference) is missing javadoc

Fixed.

- OutputExtentField(FieldReference) is missing javadoc

Fixed.

- AbstractExtentParameter
  - the header and class javadoc exceed the max line length (98)
  - if the getVariable APIs are not needed, remove them

Fixed.

> - assign - why not throw a NullPointerException? Also, the message at line 194 has problems: you have tabs and the starting quote is in the wrong place.

I copied the throw statement together with other code from OutputExtentParameter. The exception is there to catch the illegal state, without it, the error would be silently ignored. I would rather prefer IllegalStateException otherwise I have nothing against.

I am not sure which message you mean.

> - javadoc lines 217 and 234 have link out-of-place.

I am afraid I didn't localize this either.

> About your questions:
>
> > 1. OutputExtentParameter.validateAssignment raises an error of "Extent parameter dimension of %d from procedure %s is mismatched with...". I haven't figured out how to retrieve the sub-procedure name.
>
> This is a tricky one. The assignment takes place AFTER the procedure/function has already finished - thus it is no longer on the stack. I think you need to modify the APIs (starting with OutputParameterAssigner.processAssignments to pass as parameter the function/internal proc name and the external program relative name (as returned by ProcedureManager.getRelativeName).

Ok, will do.

> > 2. In the converted code, the procedure calls are generated with ErrorManager.silentErrorEnable(). This prevents some errors which are expected in some use cases, like 'Extent parameter dimension of 2 from procedure...'.
>
> Please tell me which program I can look at, to test this. In normal circumstances, the NO-ERROR clause (which is the reason why the ControlFlowOps.invoke call is enclosed in ErrorManager.silentErrorEnable/Disable) will prevent the enclosed code to raise/show errors. But there might be some exception in this case.

I am attaching a test case covering this and also the converted code, see the attached silent_error.p and SilentError.java. The problem is with calling a procedure with the no-error modifier. In Progress the error 'Extent parameter dimension of 2 from procedure...' is displayed vs. it is not in P2J.

> 3. How to detect ERROR RETURN? BlockManager.returnError doesn't seem to set a pending error and then AbstractExtentParam.assign performs the variable assignment when it should not.

> Isn't RETURN ERROR the same as raising a condition in the procedure/function?

I am not sure. BlockManager.returnError throws ReturnUnwindException which is handled before OutputParameterAssigner.processAssignments gets called by BlockManager. At the time processAssignments is called, there seems to be no flag indicating the error.

> If so, then I think if you catch the ErrorConditionException (or catch RuntimeException and check the causes), you can assume the proc/function did not end properly. If this doesn't work, you will need to set some flag in BlockManager with the state of how the last func/proc call ended - via normally, via return error, with a condition, etc.

The thrown exception is handled before the control is returned to OutputExtentParameter. I will have to investigate the possible solution closely. For example there seems to be some kind of mechanism to abort the parameter assignment through a call to OutputParameterAssigner.abort, but in this case it is not executed - either it is meant for different use case or it doesn't work as expected.

I am attaching hc_upd20140619a.zip with fixes mentioned above.

#39 - 06/20/2014 07:00 AM - Constantin Asofiei

> I copied the throw statement together with other code from OutputExtentParameter. The exception is there to catch the illegal state, without it, the error would be silently ignored. I would rather prefer IllegalStateException otherwise I have nothing against.

OK, change it into IllegalStateException.

> - javadoc lines 217 and 234 have link out-of-place.

I am afraid I didn't localize this either.

I'm talking about javadoc like this:

```
    * To indicate a failed validation, indicate an error with help of
    * @link {@link ErrorManager}.
```

I am attaching a test case covering this and also the converted code, see the attached silent_error.p and SilentError.java. The problem is with calling a procedure with the no-error modifier. In Progress the error 'Extent parameter dimension of 2 from procedure...' is displayed vs. it is not in P2J.

OK, I think I know what's happening. When you use the NO-ERROR clause, check the ERROR-STATUS:ERROR, ERROR-STATUS:NUM-MESSAGES and ERROR-STATUS:GET-MESSAGE, for details about any logged errors.

In your case, I think the following is happening:
- the error message is written directly to the terminal, regardless if the NO-ERROR clause is present or not.
- after this, an ERROR condition is raised.

To simulate this in P2J:

- when NO-ERROR is used (ErrorManager.isSilent() needs to be true):
  - use ErrorManager.displayError to show the error message to the terminal, in P2J.
  - use ErrorManager.setPending to set the ERROR-STATUS:ERROR flag
- when NO-ERROR is not used, use ErrorManager.recordOrThrowError

I am not sure. BlockManager.returnError throws ReturnUnwindException which is handled before OutputParameterAssigner.processAssignments gets called by BlockManager. At the time processAssignments is called, there seems to be no flag indicating the error.

ReturnUnwindException is used for normal RETURN processing. When RETURN ERROR is involved, an ERROR condition is raised, but only for procedures (BlockManager.returnWorker:7405):

```
      // functions don't raise error in the calling scope
      if (rtype == ReturnType.ERROR && !inFunc)
      {
         triggerErrorInCaller();
      }
```

I think this code in BlockManager.topLevelBlock is executed too late:

```
      // process assign-backs to database fields from output parameters, if
      // any
      if (opa != null &&
          (btype == BlockType.EXTERNAL_PROC ||
           btype == BlockType.INTERNAL_PROC))
      {
         opa.processAssignments();
      }
```

It might need to be executed in the finally block, which can check if an ErrorConditionException was caught or not (at line 6865).

#40 - 06/30/2014 07:04 PM - Hynek Cihlar

- File hc_upd20140630a.zip added

The attached set of changes resolves issues mentioned above. Please review.

Note that there are two non-trivial unresolved limitations.

(1) Decimal handling doesn't work exactly as expected for extent fields.

```
def temp-table tt
   field f as decimal decimals 3 extent 2.

create tt.
find first tt.

function foo returns int extent (input-output p as decimal extent).
  p[1] = 1.5555555555.
end.

foo(input-output f).
if f[1] <> 1.5555555555 then message "ERR f[1] <> 1.5555555555".
```

After the call to foo, f is expected to contain the value 1.5555555555, however in P2J it will hold the value 1.556. For regular variables this works by setting the variable temporarily to 1.5555555555. This unfortunately doesn't work for DMOs, the decimal value is rounded right away when set to the instance.

Please advise, (a) fix this as part of this issue or (b) open new one?

(2) When ControlFlowOps is performing a call and error is raised before execution is handed over to the called function/procedure block, extent wrappers are left in inconsistent state. AbstractExtentParameter invokes a registration with TransactionManager with the call to TransactionManager.registerOutputParameterAssigner(). When this registration is not paired with a "deregistration" (by calling TransactionManager.abort() or TransactionManager.processAssignments from BlockManager) all the next attempts to register again will fail.

This is due to the way extent parameters have been designed. I haven't spent much time on coming up with a solution but a possible fix seems to be to implement the registration/deregistration flow using the scope start/finished notifications. That would provide the "atomicity" to the execution of the registration methods.

#41 - 07/01/2014 05:42 AM - Constantin Asofiei

Hynek Cihlar wrote:

> The attached set of changes resolves issues mentioned above. Please review.

- the implements clause in ProcedureManager$CalleeInfoImpl needs to be on its own line.
- after renaming ProcedureManager$CalleeInfo to ProcedureManager$CalleeInfoImpl some method/ctor parameters are no longer column-aligned - please check these.

> Please advise, (a) fix this as part of this issue or (b) open new one?

I think is best to create a new one. You are suggesting that the field's decimal option can be changed by the runtime, correct?

> (2) When ControlFlowOps is performing a call and error is raised before execution is handed over to the called function/procedure block, extent wrappers are left in inconsistent state.

I think there is one other problem here. Note how TransactionManager.registerOutputParameterAssigner is called - this call originates when the i.e. OutputExtentParameter is instantiated, right? And this instantiation is done BEFORE the BlockManager.functionBlock is called... thus the wa.blocks.peek() will return the previous block (the caller), not the callee. Thus, go ahead with your scope start/finished notifications solution.

#42 - 07/01/2014 08:24 AM - Hynek Cihlar

Constantin Asofiei wrote:

> Hynek Cihlar wrote:
>
> > The attached set of changes resolves issues mentioned above. Please review.
>
> - the implements clause in ProcedureManager$CalleeInfoImpl needs to be on its own line.
> - after renaming ProcedureManager$CalleeInfo to ProcedureManager$CalleeInfoImpl some method/ctor parameters are no longer column-aligned - please check these.

Thanks, this is fixed.

> Please advise, (a) fix this as part of this issue or (b) open new one?

> I think is best to create a new one. You are suggesting that the field's decimal option can be changed by the runtime, correct?

The decimal option (defined in the field/variable declaration) stays the same, only the value doesn't honour it. In particular this happens when the value of an output or input-output parameter is assigned back to the field or variable in the calling block - the field or variable will hold the value of the default precision (10 decimal places) until it is set with new value directly (the field/variable being part of an expression as lvalue).

> (2) When ControlFlowOps is performing a call and error is raised before execution is handed over to the called function/procedure block, extent wrappers are left in inconsistent state.

> I think there is one other problem here. Note how TransactionManager.registerOutputParameterAssigner is called - this call originates when the i.e. OutputExtentParameter is instantiated, right? And this instantiation is done BEFORE the BlockManager.functionBlock is called... thus the wa.blocks.peek() will return the previous block (the caller), not the callee.

Actually this works, because wa.blocks.peek() (called by TransactionManager.deregisterOutputParameterAssigner()) is executed before new block is pushed on the stack.

Anyway, I am wondering why is this implemented this way. Why are the methods TransactionManager.registerOutputParameterAssigner and TransactionManager.deregisterOutputParameterAssigner() needed when the more generic notification mechanism exists. Even more puzzling is the fact that AbstractExtentParameter.Context does implement Scopeable but is not properly registered and so scopeStart and scopeFinished are not called.

> Thus, go ahead with your scope start/finished notifications solution.

Ok, I will fix it as part of this issue.

#43 - 07/08/2014 08:12 PM - Hynek Cihlar

- File hc_upd20140708a.zip added

The attached change set provides the following in addition to the previous change set:

- fixed the formatting issues
- fix of #2292
- fixed the potential (and likely to occur) runtime state corruption when procedure/function with extent parameter invoked through ControlFlowOps - scope management of the extent wrapper classes is now implemented through Scopeable instead of the original OutputParameterAssigner. Note that the same runtime corruption may happen with FieldAssigner as it uses OutputParameterAssigner the same way the extent wrapper classes used to.

Please review.

#44 - 07/09/2014 03:53 AM - Constantin Asofiei

Hynek Cihlar wrote:

> Please review.

The changes look good. Go ahead with conversion and runtime testing.

#45 - 07/12/2014 06:23 PM - Hynek Cihlar

Conversion and runtime tests passed. During conversion test no differences in generated code were reported.

#46 - 07/13/2014 10:33 AM - Greg Shah

If I understand correctly, no merge is needed and you are ready to check it in.

You can go ahead with the check in and distribution.

#47 - 07/13/2014 03:33 PM - Hynek Cihlar

- File hc_upd20140713a.zip added

The attached 0713a is 0710a plus up to date bzr check-ins. 0713a checked in to bzr revision 10568.

#48 - 07/13/2014 03:36 PM - Hynek Cihlar

- % Done changed from 0 to 100

#49 - 07/13/2014 05:49 PM - Greg Shah

- Status changed from WIP to Closed

#50 - 07/16/2014 05:37 PM - Hynek Cihlar

- File hc_upd20140715a.zip added

The attached hc_upd20140715a.zip resolves the issue related to the conversion of non-extent table fields passed to output or input-output

parameters. The fix has passed regression testing with no differences in the generated code. The customer server conversion test is in progress.

Please review.

#51 - 07/17/2014 02:20 AM - Constantin Asofiei

Hynek Cihlar wrote:

> The attached hc_upd20140715a.zip resolves the issue related to the conversion of non-extent table fields passed to output or input-output parameters. The fix has passed regression testing with no differences in the generated code. The customer server conversion test is in progress.
>
> Please review.

The change looks OK.

#52 - 07/19/2014 06:08 PM - Hynek Cihlar

0715a passed the conversion test on the customer's server project - there were no differences in the generated code. It was committed to bzr repository as revision 10577.

#53 - 11/16/2016 12:06 PM - Greg Shah

- Target version changed from Milestone 11 to Cleanup and Stablization for Server Features

Files

FixingOutputParam.java	1.6 KB	04/25/2014	Hynek Cihlar
hc_upd20140618a.zip	124 KB	06/18/2014	Hynek Cihlar
silent_error.p	1.69 KB	06/19/2014	Hynek Cihlar
SilentError.java	5.97 KB	06/19/2014	Hynek Cihlar
hc_upd20140619a.zip	124 KB	06/19/2014	Hynek Cihlar
hc_upd20140630a.zip	173 KB	06/30/2014	Hynek Cihlar
hc_upd20140708a.zip	173 KB	07/09/2014	Hynek Cihlar
hc_upd20140713a.zip	173 KB	07/13/2014	Hynek Cihlar
hc_upd20140715a.zip	173 KB	07/16/2014	Hynek Cihlar